Volume 2 Issue 9 $1.75 FEBRUARY 1987

"Serving 99'ers Since 1984"

THE SMART PROGRAMMER

ANNOUNCEMENTS

by Richard M. Mitchell

Bytemaster has many new plans for 1988.

First, I'm pleased to announce the MASTER series of software. The first product, STRING MASTER, was released at the 1987 TI-Faire in Chicago. At least two more products are planned for the series in '88. The new products in the MASTER series will be developed by independent programmers for release under the Bytemaster label. Bytemaster will provide input for program design, but will play a minimal role in actual development. Some of the products in the series will run under the STRING MASTER environment, while others will be stand-alone packages with options included for interfacing to other MASTER series packages. Some of the MASTER series products will be for general usage, while others will be geared toward programmers. I'm really excited about what's in the works!

Also, Bytemaster is now a dealer for a number of products and will expand into other lines as interest merits. In this issue, you will find order information for products from Genial Computerware and Disk Only Software. And, of course, Bytemaster products are listed in the catalog.

Effective immediately, all orders will be processed within 10 days of receipt. Thanks to a new order entry system, orders received in the past few weeks have been going out in 1 to 5 days.

XB XMLLNK: GET STRING SPACE

by Richard M. Mitchell

A lot of readers have been asking about using XMLLNK's. In the December '86 issue, Tom Freeman showed how to use the XMLLNK for SCROLL. In the January '87 issue, I showed an example using CFI, convert floating point to integer. Unfortunately, I must have been asleep while writing that article, as I omitted DEF START and END and improperly used CI, the occurrences of which should be changed to CLR R2, C *R3,R2 and LI R2,>20, C *R3,R2, respectively. Anyway, in this article we'll look at the XB GET STRING SPACE (note that in BASIC, GET STRING SPACE is accessed with GPLLNK).

GET STRING SPACE is accessed by EQUating GETSTR to >02 (E/A manual, p. 416), EQUating XMLLNK to >2018, loading the word >830C and >830D with the number of bytes to assign and then coding as follows:

```
BLWP @XMLLNK
DATA GETSTR
```

Well, the above is as defined in the E/A manual. Unfortunately, the E/A manual (yes, again!) does not tell the entire story. TI apparently assumed that the word "string" would be interpreted as used elsewhere, as up to 255 characters. Many users assumed that the routine was word-oriented since a word was to be loaded with the number of bytes to assign. Well, it's not! The routine is byte-oriented and therefore only does error checking for up to 255 bytes! Attempts to assign more than 255 bytes will result in a more-or-less permanent

assignment. Such assignments are not fully covered by error checking and can result in a VDP software crash! Beautiful colors, but nothing useful!

Also, do note that a garbage collection would likely affect an assigned space, so I suggest using an assigned space immediately. Perhaps garbage collections might be an interesting topic for a future article.

If the above cautions are observed, then GET STRING SPACE can be used as described on page 253 of the E/A manual, for purposes such as assuring an assigned area for a PAB or PAB data buffer. The assigned space is pointed to by the word at >831C.

SHOW NEWS

by Richard M. Mitchell

Chicago Hightlights

Chicago was once again a successful meeting ground for 99'ers in '87. Below are descriptions of some of the new items not mentioned in previous issues.

Ryte Data's Bruce Ryan was showing a new expansion box, the "99AT", that he has been busy designing. The box has 5 card slots and has a standard power supply that is rated at about twice that of a TI box (yes, full power drives!). And, Bruce says he can accomodate requests for other power supply ratings. The box accomodates up to 4 internal drives. It really looks nice. Contact Ryte Data, 21Ø Mountain Street, Haliburton, Ontario, Canada KØM 1SØ.

Steve Karasek, 855 Diversey Dr., St. Louis, MO 63126, offered SUPERBASIC, a program featuring DOS-type commands, XB editing functions, and 32 programmable keys.

Genial Computerware's J. Peter Hoddie introduced new software from his firm and was again a well-received featured speaker. Mike Dodd's PC-Transfer (Genial), listed in the Bytemaster catalog in this issue, was one of the most popular new software releases of the show.

Others participating in Chicago included Asgard Software; B & D Computer Supplies; Bud Mills Services; Bytemaster Computer Services; C & G Drives; Central Ohio 99ers; Channel 99 Users' Group; Chicago Area TI-99/4A U.G.; Chicago B128 Users' Group; Competition Computer Products; Compuserve/TI-FORUM; Corporate Disk Company; DataBioTics; Data Systems; DIJIT Systems; Disk Movers; Diskmasters; Fox Valley Users' Group; Boston Computer Society; GEnie; Great Lakes Software; Horizon Computer Limited; Hunter Electronics; Inscebot; L. L. Conner Enterprise; MYARC, Inc.; Queen Anne Computer Shoppe; Rave 99 Co.; Service Solutions, Inc.; T.A.P.E., LTD; Texaments; Tomputer Software; Trio + Software; and Will County Users' Group. And, there were at least 5 99'ers from Europe attending!

Show Calendar

TI XPO 88, Palace Station Hotel & Casino, Las Vegas, NV. February 27 & 28, 1988. Speakers: Regena, Jack Riley (MYARC). Contact John Martin (after 4 pm PST) (7Ø2) 647-1Ø62, Bob Bieber (7Ø2) 878-3167, BBS (3ØØ/12ØØ bps, 24 hrs) (7Ø2) 648-1247. Tickets: Advance, $3, Door, $5, Hotel guests $1 discount.

New England Fayuh, Diamond Middle School, Lexington, MA. April 9, 1988. Contact Walt Howe at (617) 692-27Ø2 or on GEnie WALT.HOWE or on CIS 7Ø277,353Ø or on Source TI3854. Or, write Boston Computer Society, TI-99/4A UG, One Center Plaza, Boston, MA Ø21Ø8.

TICOFF, Roselle Park High School, Roselle Park, NJ. March 26, 1988.

Canadian TI FEST, Merivale High School, Nepean, Ontario (near Ottawa), March 5, 1988. Contact Jane LaFlamme, (613) 837-1719 (home) or (613) 745-2225 (work).

9640 PROGRAMMING NOTES

Now that the MYARC 964Ø computer has finally arrived on the scene, there are many programming issues which must be addressed. There are some issues of compatibility that software authors must consider so that their programs can work on both the 99/4A and the 964Ø. There is also a whole new set of rules that programmers must concern themselves with when writing software for the 964Ø. This discussion will primarily be in reference to the 99/4A mode of the 964Ø, as at this time most development is

still taking place in 99/4A mode. Future articles will be presented on the details of the 9640's native mode, which is considerably less restricting than 99/4A mode.

CPU PAGING AND MEMORY USAGE

One of the most often asked questions about the 9640 is, "It has 512K of CPU memory, but is it accessed?" The answer is yes, through a paging scheme that is significantly more advanced than the methods used for RAM disks and the like on the 99/4A. The 9995 has a 64K address space just like the 9900, so to access more memory some sort of paging is required. MYARC chose to do the paging in 8K blocks because the architecture of the 99/4A was essentially made up of separate 8K blocks. Thus, there are 8 page spaces on the 9640, numbered from 0 to 7. The 9640 is capable of addressing up to 2 megabytes of memory (2048K of memory). If this 2 megs of available memory is divided into 8K pages, there are a total of 256 pages available. Conveniently enough, values from 0 to 255 can be held in a single byte of memory. The 9640 has 8 page registers, each one byte long. Each register controls a separate 8K page. By placing a page number into a map register, that particular page of memory will be mapped into that page in the 64K address space. Figure 1 is a listing of what memory space is controlled by each map register. The map registers can be read as well as written to and behave as regular memory locations, thanks to the magic of the 9640's gate array. For example, to put page >45 in the >E000 space, the assembly programmer could use the following lines:

```
LI   R0,>4500
MOVB R0,@>800
```

The map registers may also be accessed with the MOV (move word) instruction, so that the entire map can be saved or loaded in 4 instructions.

Pages >00 to >3F are contained on the 9640 card itself. This accounts for the 512K of CPU memory that is available. In 99/4A mode, page >03 must be mapped in at location >C000 (map register 6) for GROM/GRAM and sound to be present. If a page other than >03 is present, the 9640 will essentially ignore any memory mapped I/O to the GROM and sound ports. Also, in 99/4A mode the map register for the >6000 memory space (register 3) can be changed; however, this will have no effect, since this page is locked as >36 by the gate array. If 2 banks of cartridge RAM are in use (such as in TI Extended BASIC), the second page is >37. Because the 9640 emulates GROM/GRAM using CPU memory, 8 pages of memory are also reserved for this purpose. The pages >38 to >3F are the GROM pages. These pages are offset by one byte. For example, GROM byte 0 is found at location >0001 on page >38 and GROM byte >FFFF is found at location >0000 on page >38.

On the 9640 computer, there is also 32K of high speed, zero wait-state CPU memory. This is divided into 4 pages, numbered >EC, >ED, >EE, and >EF. If one or more of these pages is available, they should be used for time-critical portions of code, since they are noticebly faster than the regular pages. The EPROM that boots the 9640 (including the swan picture) is 16K long, occupying 2 pages, >F8 and >F9. If the 9640 is running with a modified MYARC 512K card, it will appear as pages >80 to >BF.

There are 8 more special pages in the mapper to be considered. These pages are >B8 to >BF, and when they are accessed the

Register number	First Address	End Address	Map Register Address
0	>0000	>1FFF	>8000
1	>2000	>3FFF	>8001
2	>4000	>5FFF	>8002
3	>6000	>7FFF	>8003
4	>8000	>9FFF	>8004
5	>A000	>BFFF	>8005
6	>C000	>DFFF	>8006
7	>E000	>FFFF	>8007

Figure 1

memory cycles are passed to the rest of the expansion box. At present, the only really useful application of this is to access memory mapped I/O in the DSR space. This is done using page >BA (the third bus page, corresponding to addresses >4000 to >5FFF). For example, if page >BA was placed in map register 2 (the >4000 page) and the TI RS232 card was turned on (using a LI R12,>1300 followed by a SBO 0 instruction) a MOVB instruction to address >5800 would write a byte to the parallel port. However, if page >BA was placed in map register 0, then writing to address >1800 would accomplish the same. Since the 9640 has all its own DSR's (that is, it ignores the ROM's in peripherals), a page other than >BA is mapped in the >4000 space. This means that print spooler software that directly accesses the PIO register (for the MYARC and TI RS232 cards) will no longer work. This can be easily fixed by having these programs put page >BA in the appropriate map register before writing to the parallel port. However, the program must be careful to restore the proper page when it is finished or the system may not behave correctly when an interrupt occurs, a peripheral access is attempted, or a software reset occurs.

One difference between the 9640 and the 99/4A is that the 9640 has RAM from >8020 to >82FF whereas the 99/4A has shadows of the >8300 to >83FF memory in that area. A few programmers accessed the >8300 page through one of its shadows at >8000, >8100, or >8200 on the 99/4A. Most of these programs will not run on the 9640. However, when patching 99/4A software on the 9640, having an extra 700 bytes or so of space that didn't exist on the 99/4A can be very useful. Another related difference is that on the 9640 the RAM at >8300 is not any faster then the rest of memory. On the 9640 the fast RAM is from >F000 to >F0FF. When writing assembly code for the 9640, if at all possible put the registers at >F000 for maximum speed. There is one problem when using the >F000 to >F0FF space, and that relates to the 9995. The fast memory at >F000 is actually in the 9995 microprocessor, which means that it can never be paged out. If the >E000 space is changed using the map register, the bytes from >F000 to >F0FF will remain the same. However, there is one more catch. If a write is performed to memory in the >F000 to >F0FF range, the memory in the page behind it will be trashed. The reasons for this have to do with memory speed. However, the point is that it is best not to change the >E000 space page. For example, in My-Word the >E000 space is used primarily for the code which is responsible for switching between the Editor, Formatter, Catalog, and Help portions of the program, so the >E000 page itself never changes.

VDP PAGING AND MEMORY USAGE

The 9640 contains 128K of video memory, whereas the 99/4A contains only 16K. Those familiar with how routines like VSBW and VMBR actually work (rather than just how to use them) will quickly recognize that some sort of VDP paging must exist in order to access all 128K of memory. However, VDP paging is handled quite differently from CPU paging. The 128K of video memory is divided into 8 pages of 16K. There is one VDP page register, and it is VDP register 14. To select a VDP page, the page number must be set in this register. In the example below the VWTR (VDP Write to Register) routine is used to select page 3:

```
LI   R0,>0E03
BLWP @VWTR
```

On the 99/4A, when VDP address >3FFF was accessed the VDP address automatically wrapped back to >0000. On the 9640, the situation is the same when working in any graphics mode that was present on the 99/4A. However, when working in a 9640 graphics mode, the page register will increment by 1, and the address will increment to >0000 on the next page. Thus, when working in a 9640 graphics mode it is possible to read/write up to 128K of data without resetting the VDP address.

To display screens stored on VDP pages other than zero, information must be passed to the various VDP registers to specify which pages. Such information is beyond the scope of this article. It is covered in the 9938 manual, which is available from Yamaha (with a little luck) or from MYARC for $20. Those planning on doing any serious work for the 9640 should obtain a copy of this manual which describes all the capabilities of the 9938 graphics chip. Perhaps a future series of articles will explore this chip in detail.

OTHER CONSIDERATIONS

Because the 9640 has such a radically improved keyboard from the 99/4A, the console KSCAN (keyboard scanning) routine

was completely changed. From a functional perspective, the routine in the 9640 is identical to that in the 99/4A; however, in terms of how they work, nothing is the same. This means that any 99/4A program that scanned the keyboard without using the console KSCAN routine will not respond to keyboard input on the 9640. In general, these programs fall into 2 catagories: (1) terminal emulators which couldn't use the console KSCAN because it was too slow; (2) interrupt routines which would check to see if a weird key combination such as Control-Shift-something or Function-Shift-whatever was pressed to invoke special commands. If the only reason for not using the console KSCAN routine was for speed, then programs can be easily converted by the author to run on the 9640. The program simply has to check to see if it is running on a 9640 (RAM at >0000 is a good check) and if so, use the console KSCAN. If the reason for avoiding the console KSCAN was for weird key press combinations, then more drastic changes may be required by the author.

The 99/4A emulator for the 9640 is capable of running at 5 speeds, with speed 5 being the fastest. The only factor that keeps many 99/4A programs from running at speed 5 is related to VDP access. TI stated that in setting the VDP read or write address, that it was necessary to wait after writing each byte of the address. On the 99/4A, because of the many wait states inserted by the computer, there was really no need for this. On the 9640, running at faster speeds, these delays are required. In general, it has been found that using a SWPB (swap byte) instruction after each byte will work. A NOP (no-operation) or RT (return) is generally too fast. One known offender is the GPL/DSRLNK published in *The Smart Programmer* by Craig Miller and Doug Warren. In a related note, TI also said that is was necessary to insert delays when accessing GROM/GRAM. This is not so on the 9640 because GROM/GRAM is handled by the gate array without any delays. *(Editor: The GPL/DSRLNK routine was optimized for the 99/4A, conforming with TI's practices rather than TI's suggestions. RM)*

Those familiar with the MYARC hard disk personality card are probably aware that it is capable of transferring data directly to CPU memory rather then passing all data through VDP memory. On the 9640, this capability has been expanded to include all DSR calls. Note that the PAB must still reside in VDP memory, regardless of where the data is passed. To indicate that data is to be transferred through CPU memory, set the 4 bit of the high nibble of the op-code when doing a DSRLNK. For example, to read to CPU memory, use a read opcode of >42 instead of >02.

When programming for the 9938 video chip, be careful to consider unused bits in the VDP registers. Some bits that were not used on the 9918A are used on the 9938 (rumor has it that some versions of Forth set certain unused VDP bits to 1 that the 9640 expects to be 0, which can produce weird video displays). For example, the high bit of VDP register 1 tells the 9918A whether 4K or 16K of video memory is available. This bit is not used on the 9938. Setting this bit to 0 when writing on the 9640 will cause the program to bomb when running a 99/4A. When working in the new video modes (like 80 columns), it is important to set all the bits to 1 that the manual specifies, even if it doesn't make sense. Failing to set certain bits can produce some fascinating results.

5th 1- =FORTH

by Mariusz Stanczak

Last time we met, I promised to introduce you to the brave world of "artificial intelligence" (AI), on the road to which goal we will embark in this installment of 5th 1- =FORTH, in addition to having a mini review of a product that was "dropped at my door" for no other apparent reason than to be mentioned in this column, and, at least, a mention it certainly deserves. *(Editor: Due to the length and significance of this article, the final installment of PSEUDO83 has been postponed. RM)*

The product is a wonderful collection of functionality and innovation executed with the best of tastes, and it is collectively called Super Space. It is so feature-packed that I'm having a problem with where to start, but a reasonable way, if not the only possible given the number of products included in the package, would be to "divide and conquer", so that's what I'll try, my only hesitation being that the total of the package is much greater than the sum of the parts, but I hope that this will become apparent as we go along.

First of all, Super Space includes the hardware part, which consists of a standard cartridge module. Give away your E/A module, as that's included in the Super Space, but there's more. In addition to the standard TI E/A GROM, there is either 8KB (Super Space I), or a whopping 32KB (Super Space II) of battery backed RAM. Brevity is the aim of this column, so I will just say that all the hardware's what's and how's are extensively documented in the supplied manual. The software's side can be described as simple, useful, and flexible. Among the utilities are menu builder, loader, and editor, which program permits creation of a custom opening menu (after the power-up screen). The menu, which is permanently (subject to well documented precautions, although I had no problems whatsoever despite my purposeful violation of them all) retained in the battery backed RAM of the Super Space, then will auto-load and run, from any disk, any one of the seven program entries that can be created. Another program, CVAC, permits saving to, and later loading and executing from, disk of cartrige ROMs (8KB, none-paged only). Yet, another program, SSLDR (Software Support LoaDeR), which provides a 1Ø entry menu, can be loaded into Super Space. Each of the entries provides for an auto-loading option designed to make usage of our venerable consoles in a more efficient way. There are entries to load Editor, Formatter, and Program Editor (modified versions of TI-Writer and Formatter are supplied), an entry to load assembler (a Macro Assembler from R.A.Green, a fine Fairware program is supplied), and one to load a standard Utility program (any program named UTIL1, for example a spelling checker for TI-Writer). Then, there are entries to Print a file from disk, to Configure a printer (printer parameters can be changed dynamically by this option), and to do a Disk Directory (yes, without loading of your favorite disk manager cartridge!, an option which is also provided in the supplied versions of editors). Altogether, two packed "flippies" worth of utilities and tools, but as if that were not enough, the Super Space II package also adds another "flippy" with Mr. Clint Pulley's Fairware C99 compiler and two books (Programs for the TI Home Computer, and Introduction to Assembly Language for the TI Home Computer) on programing. Well (after catching my breath), what more can I say? Oh yes! The company is called DataBioTics Inc. and can be contacted by writing to PO Box 1194, Palos Verdes Estates, CA 9Ø274.

By now, you might be asking what all the above has to do with our column on Forth, and why Super Space hasn't been described to you by Richard in his product reports in *TSP*? But then, there is something I haven't mentioned yet; namely, that in the package sent to me there was also another product, a product tailored to work with the Super Space of which environment's features it takes advantage (read, Super Space is a "must have" prerequisite to it), but that it is purchased separately (also from DataBioTics). This product is called Super 4TH, and I could only hope I could get as excited about it as I did about the Super Space package. Not that it is a bad product (in which case I wouldn't be writing about it), but neither can I find in it much innovation, besides the very initiative of DataBioTics to support it and to supervise its further development. Super 4TH intends to be the evolution of TI-Forth and version 1.1 furthers that goal by a rather insignificant step. It is a "bug-fix" update of the public domian TI-Forth additionally supported by much of what had been written by the many "nameless" and generous Forth enthusiasts in the TI and Forth community at large. As a collection of those tools, Super 4TH is a perfect buy for the latecomers to Forth or to those users of Forth who haven't done much snooping around in Forth related publications in the quest for their own tools.

Having said that, let's have a quick look at the scope of changes Super 4TH brings into the TI-Forth environment. The range goes all the way from "simple" bug fixes through some new features. On the "new" side of things, there is the addition of an auto-repeat cursor in the 4Ø-column editor (but not on the command line; i.e., not in the "immediate", interpretative mode of the system), and there is a set of graphics characters (re)defined, which allow for graphics without the need of switching into bitmap mode. Also provided is a set of words that permit usage of hard disk for screen saving and loading in Super 4TH. In the general system improvements category are: loading part of the system's kernel into the Super Space memory space where it is retained and later available upon power-up, and having resident system messages upon loading, and the availability of an extra disk buffer (six total, as opposed to five in TI-Forth). There is another improvement claimed that would belong to

this category, that is, optimization of the kernel code for both speed and space improvement, and though the space optimization is hard to verify (the system takes less of the 32KB total RAM available to TI-Forth, but it adds some new words, and redefinitions of existing ones, in additon to using the extra 8KB of the Super Space), the speed "improvement" in the tests I ran with "Forth Dimension benchmarks" was about -1Ø% (yes, minus), which I found a curious, but insignificant change. In general, the system has a nice feel of a well setup TI-Forth to it, with its "image" touched-up, utilities collected in functionally well thought out blocks of BSAVED code (with acompanying source blocks supplied on the 'flippy' side of the system disk), and a (A5 format) manual that seems to be as extensive as TI's, so if you're considering the Super Space package and company support for what you buy is of importance to you (DataBioTics sounds serious on this point), maybe then you might also find Super 4TH a worthwhile addition to your system.

And speaking of worthwhile additions, let's switch our attention now to the "main course" in our quest for "roll your own" Forth.

The term AI, in my modest opinion, not only became the most "in" buzzword of the past few years, mostly for salesmen, but it also is wearing thin, if one considers the products that this phrase is applied to, and applied it is quite liberally. But not all is lost and, indeed, great strides have been made in the evolution and application of AI, and one of the oldest areas to which AI has been applied with great success is in the expert systems field. The aim of an expert system is to capture the knowledge of a specialist in a given field, after which the computer is able to react, in an intelligent way, to a given set of circumstances, as they are presented to the computer in data fed into it; not unlike the specialist whose knowledge now had been "taught" to the machine. The fundamental vehicle and a programming methodology for delivering that knowledge to the computer is a "production system", which is so called not because it is used to produce an expert system, but because it uses pairs of conditional IF-THEN statements to describe the knowledge of a specialist and each of those pairs is called a rule, or a production. In addition to rules, production systems have a way of representing "facts" to which the productions are applied in the process of finding a solution, and the mechanism which performs this application is called an "inference engine" and lets only say that there are two major ways by which the inference engine may tackle a problem: the so-called backward chaining in which finding a solution is goal driven; i.e., to a presented goal the system will try to match to it the action portion of a rule, and if the match is found, the condition portion of that rule becomes the new subgoal etc. until all goals are satisfied (success), or no rule is found that matches the current goal (failure); and the so-called forward chaining method which is data driven; i.e., for a given set of facts, the system will attempt to find a match among the conditional part of rules, and if the match is found, the action portion of the rule will be executed, and the process terminates when no condition matches given fact(s), or an explicit HALT command is executed as part of rule's action. Let's leave this short introduction to production systems at that, as the specifics get rather involved, and of not much interest to a user (teaching expert) of such a system, which, I hope, you become. Production systems, because of the amount of procesing that they, in general, require (they are mostly programmed in LISP which pays a heavy price in performance for its flexibility), come with dedicated hardware (TI-explorer, Symbolics, LMI) and that makes them quite expensive, which still holds true even when they are delivered as software-only (OPS5) solutions. Well, not in our case, and do not think then that it must be something greatly inferior. FORPS (FORth Production System) as is presented on these pages, was developed at Oak Ridge National Laboratory, which is operated by Martin Marietta Energy Systems, and designed to be a complete production system for real-time applications. A number of "serious" applications have already been written in FORPS, and the system has been ported to the Novix Forth chip (on which it runs circles around the dedicated AI workstations). The design philosophy, according to the author of the system (Matheus, C.J.), was driven by three main goals:

- to "provide the representative power of a production system"
- to "maintain the advantages of Forth (including its extensibility and its interpretative nature) and"
- to maintain "speeds appropriate for many real-time applications"

In order to satisfy the above goals, FORPS does sacrifice some of the "bells and whistles" of more extensive, and slower systems, but because of its inherent extensibility, such features can be added if so it would be desired. The unique quality of FORPS, as compared to other production systems written in Forth (Expert-2, which mimics LISP), is that it is written completely in Forth; it exists as an integral part of the Forth environment, in which any Forth

word can be executed from within the FORPS rules, which themselves consist of Forth words.

In FORPS, a fact is represented by any Forth word that evaluates to a binary, truth, or falsity, value, and rule definition is modeled after a conventional production system by the provision of the mechanism: RULE: name PRIORITY: value *IF* condition *THEN* action *END* which above words constitute a complete rule specification, with priority assignment being an optional part of the specification. In FORPS, rules are stored in a table, which stores pointers (CFA's) to a rule's condition part, action part, and which also contains the active/not_active flag, and the rule's priority (if assigned, otherwise zero). The number of rules in the system is defined by a constant MAX-#RULES, and the length of each rule by another constant, RULE-LEN, so the size of the RULE-TABLE table is defined by the product of those two and ALLOTed at the time the system is compiled. The RULE-TABLE is cleared by the RESET-FORPS word, after which rules are entered into the table as described above. After the rule description is complete, the system is started by the word FORPS, which contains the system's inference engine (the portion of code between the BEGIN-UNTIL construct) which cycles through the RULE-TABLE (CLEAR-FIRES TEST-RULE-CONDS SELECT-BEST-RULE FIRE-RULE) until there is no more activity (NO-ACTIVITY @ UNTIL). It is that simple! The evaluation of rule's condition is reduced to EXECUTE'ing its CFA (remember the conditions evaulate, on the stack, to a boolean flag), and after each condition in the table is evaluated, each rule's action flag will contain either 1, or Ø, which values are then used by SELECT-BEST-RULE to find the rule with the highest priority (stored in HIGH-PRI, a variable holding the priority on the current BEST-ACTIVE-RULE), which rule becomes the BEST-ACTIVE-RULE (a variable) by having its action CFA stored in it, and consecutively EXECUTEd by FIRE-RULE. The rule definition mechanism, in addition to making an entry in the RULE-TABLE, also creates a header in the (FORTH) system dictionary very much like the : word, so each rule's condition part can be tested interactively. The structure of this entry is also alike that of any other colon defined FORTH word, with the addition of a second CFA that follows the standard structure (after the regular "exit" by compiled ;), which is the CFA of the rule's action (also terminated by the "exit" word). That is basically the structure of FORPS; simple, compact, and the whole is very fast. The code is below, followed by the Towers of Hanoi example. The interesting aspect of this example is that although FORPS is based on the backward chaining approach, the game itself is a simulation of the backward chaining method, to show that either is possible. I may also add that the algorithm implemented in the game is the same as the one used in the recursive example as presented in the last column; only the programming methodology differs. Have fun, and for more information on FORPS either check out *The Journal of Forth Application and Research* (V4/1), or the East Coast Forth BBS, which has a separate FORPS section, or drop me a line, or two, which you are welcomed to do in any Forth related problem that comes your way. 'Till the next time, Mariusz.

Listing 1, FORPS

```
\ FORPS - rule definition
10 CONSTANT MAX-#RULES
8 CONSTANT RULE-LEN
0 VARIABLE NO-ACTIVITY
0 VARIABLE 'SP-IF
0 VARIABLE 'NOOP
0 VARIABLE >LAST-RULE
0 VARIABLE >RULE-TABLE
0 VARIABLE CYCLE
0 VARIABLE HIGH-PRI
0 VARIABLE BEST-ACTIVE-RULE
0 VARIABLE RULE-TABLE  MAX-#RULES RULE-LEN * 2- ALLOT
: FALSE  ( a --)  0 SWAP ! ;
: TRUE  ( a --) -1 SWAP ! ;
: >ACTION  ( a -- a) 2+ ;
: >FIRE-CELL  ( a -- a) 2+ 2+ ;
: >PRIORITY  ( a -- a) 6 + ;
```

BYTEMASTER
SPRING 1988 CATALOG

STRING MASTER
(Bytemaster)

- Exciting user environment for running tommorrow's new generation of 4A and 9640 software
- Powerful programming tool, written in assembly language for speed
- Runs in Extended BASIC
- 29 Extended BASIC LINKs
- Windows
- Compatible with almost all XB programs and XB LINK code through selectable memory placement
- Conforms to XB standards
- Compatible with standard hardware
- Allows working with entire arrays in a single program statement
- Easy to use

Words Were Just Data Until... STRING MASTER

Many Extended BASIC (XB) programs use text strings extensively, making STRING MASTER an ideal companion to XB. STRING MASTER is an exciting new programming tool and user environment. The package consists of a set of assembly macro equivalents of XB string manipulations, in the form of LINKs to XB. STRING MASTER can be loaded into the assembly program memory area (High Mem) or the XB program area (transparently residing in Low Mem), enabling full compatability with almost all assembly LINK code and XB programs. Because many STRING MASTER functions are oriented toward string segments and concatenations ("fields" and "records"), the package is a perfect foundation for a database language. STRING MASTER fully supports every data type (direct, variable, array element, array) for both numeric and string parameters in either OPTION BASE (Ø or 1), including support of multi-dimensional arrays (up to XB's limit of 7). Working with an entire array in a single statement can greatly reduce programming effort and program execution time. For instance, STRING MASTER's peeks and pokes (CPU and VDP memory) are not limited to 1 byte or even 255 bytes, but rather can utilize an entire array of bytes, limited only by available string memory (up to about 12K bytes)! Perhaps the most dramatic feature of the package is its ability to very quickly write and read screen row, column boxes (WINDOWs) directly to and from strings or string arrays. An append function allows use of operators (>, <, =, etc.) to selectively copy array elements or segments of array elements to any point in an array and provide a count of the elements copied or even to simulate the copy for the purpose of simply obtaining the count only! Other features include search, select, replace, sounds (BEEP, HONK, REBEEP), sorts, base conversions (yes, convert an entire array in one statement!) and much more. There are 29 routines in all. Dr. Ron Albright, writing in *Computer Shopper*, called STRING MASTER "...one of the most valuable software development tools the 99/4A has seen since the Extended BASIC cartridge." Talented 4A programmer Mike Dodd says, "extraordinary.... does to arrays everything I can think of....". STRING MASTER includes thorough printed documentation and an extensive demo program. Tutorials and applications will be presented in major 99'er publications. Runs on the 99/4A or MYARC 964Ø. Requires XB (TI, Triton, Mechatronics or GK), expanded memory and disk system (cassette, inquire). Only $19.95.

MG EXPLORER (Bytemaster)

♦ Your window into the 99/4A

♦ Unlocks secrets

Bytemaster is now the exclusive distributor for this popular program. The package includes versions of the program to run from every major 4A programming environment, plus the famous EXPLORER manual, all packed onto four flippy disks. The program can now be run from your RAM disk, hard disk, Gram Kracker or any floppy drive. EXPLORER emulates system activity, displaying the information on screen when desired or displaying the actual screen of a running application. De-bug your programs, learn how XB works, discover what GROM really does, much more! Hundreds of programmers have found EXPLORER to be an incredibly effective tool. Runs on the 99/4A. Requires XB or E/A or Mini Memory or Gram Kracker or TI-Writer, memory expansion and disk system. Only $24.95.

PC-Transfer (Genial Computerware)

♦ MS-DOS, 4A, 9640 file compatability

♦ Formats MS-DOS disks

Programmed by Mike Dodd, PC-Transfer is the fastest and most convenient method available to transfer data between a 99/4A or MYARC 964Ø and an MS-DOS based machine. Simply place an MS-DOS disk in one disk drive and a TI disk in another and PC-Transfer goes to work. PC-Transfer catalogs disks and allows selection on screen of text files to copy. PC-Transfer even formats MS-DOS disks! The program also includes a hook for future implementations for other file types. Runs on the 99/4A and MYARC 964Ø. Requires either a CorComp or MYARC disk controller, two disk drives, and Extended BASIC or TI-Writer or Editor Assembler cartridge. Only $25.

Remind Me! (Genial Computerware)

♦ Manage your schedule

♦ Check your calendar

Remind Me! is a program designed to help manage a monthly schedule. Written in assembly, Remind Me! is fast! For each date, maintain up to 12 lines of text through a TI-Writer style editor. Reminders are highlighted to make checking your calendar easy. A fast search feature is included which ignores the case of letters and allows for multiple searches. Print an entire month's calendar or any range of days in a month to your printer or to disk. Choose screen colors, printer codes, printer device and more and store to disk with the program (not a cumbersome separate file). Though a clock is not required, Remind Me! will display the current time if you have a CorComp Clock or Triple Tech, MBP clock, John Clulow Clock Board, or a MYARC 964Ø. Programmed by John Johnson, author of MENU. Runs on the 99/4A or MYARC 964Ø. Requires Editor/Assembler or TI-Writer or Super Cart or Extended BASIC. Only $15.

XB: Bug (Genial Computerware)

♦ Award-Winning Program

♦ De-bug XB programs

J. Peter Hoddie's XB: Bug was overall winner of the 1st Annual TI Forum Computer Shopper Programming Contest. XB: Bug is written in assembly and resides transparently until invoked. View and modify variable values, search variables by name and by value, search the program listing for any group of characters, see the program line that is currently executing, and display all character definitions, sprite data, and color settings. Trace GOSUBs and subprogram CALLs and show which, if any, subprogram is currently active. Access the XB: Bug program with a keypress or through a breakpoint, then return to the XB program exactly where it left off. Several versions are included, allowing compatability with the memory locations of other assembly code. Runs on the 99/4A or MYARC 964Ø. Requires Extended BASIC (TI or Triton or Mechatronics or GK), expanded memory and disk system. Only $15.

XBasher (Genial Computerware)

♦ Conserve disk and memory space

♦ Load & execute programs faster

Mike Dodd's XBasher shortens variable names in XB programs, combines program lines where program logic will not be compromised, removes REM and ! statements (and even changes references to those lines), shortens the names of functions defined in DEF statements, and uses other innovative compaction methods. All compaction methods may be turned on or off to meet the user's needs. Up to a 33% compaction is not uncommon. XBasher received a straight A review from GEnie TI RT sysop Scott Darling. XBasher is ideal for anyone who wishes to decrease execution and load times of XB programs, while also saving disk space. Runs on 99/4A or MYARC 9640. Requires TI Extended BASIC, memory expansion, disk system. Only $10.

GRAM Packer (Genial Computerware)

GRAM Packer, by J. Peter Hoddie, allows customizing the main TI menu to contain programs, cartridges, and program loaders. Store multiple E/A option 5 type programs in GRAM space for near instantaneous loading from main menu, CALL statements, or even the RUN command. Additional special utilities allow assembly programs, Extended BASIC programs, and even cartridges to reside on disk, RAM disk or hard disk, but to appear on the main menu, available at a single key press. Extensive documentation is included. Requires a GRAM emulation device (tested with GRAM Kracker, GRAM Karte, Maximem, MYARC 9640 computer), expanded memory, disk system. Only $10.

Graphics Expander (Genial Computerware)

J. Peter Hoddie's Graphics Expander quickly and easily enlarges fonts, rotates characters and creates mirror or inverse images. Written in assembly for speed, the program is compatible with fonts and graphics for both TI-Artist and CSGD and can convert fonts and graphics between the two formats. All graphics are displayed on the screen and images may be scaled horizontally and/or vertically by factors of 1 to 9. Runs on the 99/4A or MYARC 9640. Requires XB or E/A or TI-Writer, expanded memory and disk system. Only $10.

XBQB (Bytemaster)

XBQB is a Quick BASIC subprogram designed to aid PC users in co-developing or cross-developing BASIC programs on the 4A and PC. XBQB emulates the XB ACCEPT AT statement. Requires IBM PC or compatible, 5 1/4" floppy drive, Quick BASIC 3.0 or 4.0. Only $1.

Orphan Survival Handbook (Disk Only Software)

Dr. Ron Albright's compilation, containing helpful recipes, assertions, advice and other nostrums for owners and custodians of the TI-99/4A and Geneve. Great information and reading. Programs, tips and much more. Only $17.

Best of Super 99 Monthly On Disk (Bytemaster)

All of the programs that appeared in Super 99 Monthly (except FORTH programs) are included, plus a few bonuses. A two disk set. Documentation is not included (refer to the articles in Super 99 Monthly). Only $12.

Super 99 Handicapper (Bytemaster)

A program to aid in thoroughbred horse race handicapping. Provides unique and useful data. Documentation with tips included. Requires disk system (2 drives recommended), expanded memory, Extended BASIC. Printer recommended. Only $15.

ITEM	EA	QTY	AMOUNT
SOFTWARE & BOOKS:			
STRING MASTER	19.95	_____	_____
EXPLORER	24.95	_____	_____
PC-Transfer	25.00	_____	_____
Remind Me!	15.00	_____	_____
XB: Bug	15.00	_____	_____
XBasher	10.00	_____	_____
GRAM Packer	10.00	_____	_____
Graphics Expander	10.00	_____	_____
XBQB	1.00	_____	_____
Orphan Surv. Hbk.	17.00	_____	_____
Best of S99M Disk	12.00	_____	_____
S99 Handicapper	15.00	_____	_____
SUBTOTAL, SOFTWARE & BOOKS			========
OVER $50? DEDUCT 10%			
OVER $150? DEDUCT 15%			
OVER $500? DEDUCT 25%			(_____)
TOTAL, SOFTWARE & BOOKS			========
BACK ISSUES:			
SUPER 99 MONTHLY	1.00	_____	_____

MONTH	QTY	MONTH	QTY
09/84	___	06/85	___
10/84	___	07/85	___
11/84	___	08/85	___
12/84	___	09/85	___
01/85	___	10/85	___
02/85	___	11/85	___
03/85	___	12/85	___
04/85	___	02/86	___
05/85	___		

01/86 (photostat)	2.00	_____	_____
TOTAL, COLUMN 1			========

ITEM	EA	QTY	AMOUNT
BACK ISSUES (CONT.):			
SMART PROGRAMMER PHOTOSTATS	2.50	_____	_____

MONTH	QTY	MONTH	QTY
02/84	___	06/84	___
03/84	___	07/84	___
04/84	___	08/84	___
05/84	___	09/84	___

ORIGINALS	1.75	_____	_____

MONTH	QTY	MONTH	QTY
06/86	___	11/86	___
07/86	___	12/86	___
08/86	___	01/87	___
09/86	___	02/87	___
10/86	___		

SUBSCRIPTIONS, SMART PROGRAMMER		
U.S. 1ST CLASS	18.00	
CANADA 1ST CLASS	20.00	
FOREIGN AIR MAIL	32.00	_____
TOTAL THIS COLUMN		======
PLUS TOTAL FROM COLUMN 1		_____
TOTAL BEFORE TAXES AND SHIPPING		======
RESIDENTS, ADD SALES TAX:		
Lake Charles, Sulphur, Iowa, DeQuincy, Vinton	3 1/2%	
Calcasieu Dist #1 only	2 1/2%	
Calcasieu Sch Brd only	1 1/2%	_____
Louisiana	4%	_____
Add: Shipping & handling, except for subscriptions		
U.S. & prot., Canada	2.00	_____
All other	8.00	_____
TOTAL REMITTANCE ENCLOSED		======

PRICES ARE IN U.S. DOLLARS. CHECKS MUST BE DRAWN IN U.S. FUNDS ON U.S. BANKS, CODED FOR PROCESSING THROUGH THE U.S. FEDERAL RESERVE BANKING SYSTEM. U.S., CANADIAN & INTERNATIONAL MONEY ORDERS ACCEPTED. NO BILLINGS OR CREDIT SALES. QUANTITY DISCOUNTS AVAILABLE. SOFTWARE LISTED IS COPYRIGHT & NOT COPY PROTECTED. 4A AND 9640 DISKS SS/SD. PERMIT BELOW IS NOT FOR REPLY MAIL.

BYTEMASTER
171 MUSTANG STREET
SULPHUR, LA 70663

NAME_____

STREET_____

CITY_____

STATE_____ZIP_____COUNTRY_____

+-----------------+
| BULK RATE |
| U.S. POSTAGE |
| PAID |
| SULPHUR, LA 70663 |
| PERMIT NO. 141 |
+-----------------+

```
17. : HALT  NO-ACTIVITY TRUE ;
18. : *ERROR*  ." No rules loaded" QUIT ;
19. : *RESET-FORPS*  RULE-TABLE DUP >RULE-TABLE ! MAX-#RULES RULE-LEN * ERASE
20.       [ ' *ERROR* CFA ] LITERAL RULE-TABLE ! ;
21. : NOOP ;
22. ' NOOP CFA DUP  'NOOP !  @ CONSTANT COLON-CFA
23. : COND-CFA!  HERE 2- >RULE-TABLE @ ! ;
24. : ACTION-CFA!  HERE 2- >RULE-TABLE @ >ACTION ! ;
25. : 'NUMBER  -FIND
26.       IF  DROP CFA EXECUTE
27.       ELSE  HERE NUMBER DROP
28.       ENDIF ;
29. : PRIORITY:  'NUMBER  >RULE-TABLE @ >PRIORITY ! ; IMMEDIATE
30. : RULE:  >RULE-TABLE @ RULE-TABLE - RULE-LEN / MAX-#RULES =
31.       IF  ." No room" QUIT  ENDIF
32.       CURRENT @ CONTEXT ! [COMPILE] : COND-CFA! SMUDGE ;
33. : *if*  -1 SP@ 2- 'SP-IF ! ;
34. : *IF*  >RULE-TABLE @ >FIRE-CELL [COMPILE] LITERAL COMPILE *if* ; IMMEDIATE
35. : *then*  -1  'SP-IF @ SP@
36.       DO  AND  2 +LOOP
37.       SWAP ! ;
38. : *THEN*  COMPILE *then*  COMPILE ;S  COLON-CFA ,  ACTION-CFA! ; IMMEDIATE
39. : *END*  RULE-LEN >RULE-TABLE +!  COMPILE ;S [COMPILE] [ ; IMMEDIATE
40. \ FORPS - inference engine
41. : SET-DEFAULT  -1 HIGH-PRI !  'NOOP BEST-ACTIVE-RULE ! ;
42. : RT-LIMITS  ( --n n)  >LAST-RULE @ RULE-TABLE ;
43. : CLEAR-FIRES  RT-LIMITS
44.       DO  0 I >FIRE-CELL !  RULE-LEN +LOOP ;
45. : TEST-RULE-CONDS  RT-LIMITS
46.       DO  I @ EXECUTE  RULE-LEN +LOOP ;
47. : SELECT-BEST-RULE  NO-ACTIVITY TRUE  SET-DEFAULT  RT-LIMITS
48.       DO  I DUP >FIRE-CELL @
49.          IF  DUP >PRIORITY @ DUP HIGH-PRI @ >
50.             IF  HIGH-PRI ! >ACTION BEST-ACTIVE-RULE !
51.                NO-ACTIVITY FALSE
52.             ELSE  DROP DROP
53.             ENDIF
54.          ELSE  DROP
55.          ENDIF  RULE-LEN
56.       +LOOP ;
57. : FIRE-RULE  BEST-ACTIVE-RULE @ @ EXECUTE ;
58. : FORPS  >RULE-TABLE @ 2- >LAST-RULE ! 0 CYCLE !
59.       BEGIN
60.         1 CYCLE +!
61.          CLEAR-FIRES
62.          TEST-RULE-CONDS
63.          SELECT-BEST-RULE
64.          FIRE-RULE
65.          NO-ACTIVITY @
66.       UNTIL  CR CYCLE ? ." Rules Fired" CR ;
67. ;S
68. FORPS is under the copyright of Martin Marietta Energy Systems.
69.    Author: Christopher J. Matheus
70.            University of Illinois
71.            222 Digital Computer Lab
72.            1304 W. Springfield Ave.
73.            Urbana, IL  61801
74. This software was developed at Oak Ridge National Laboratory,
75. Oak Ridge, TN, under the Consolidated Fuel Reprocessing Program.
76. Ported from PollyForth-79 to TI-Forth by Mariusz Stanczak.
```

Listing 2 - Towers of Hanoi in FORPS

```
 0. \ constants and variables
 1. 10 CONSTANT MAX-#DISKS
 2. 1 CONSTANT HOME
 3. 2 CONSTANT GOAL
 4. 1 CONSTANT MOVE-TOWER
 5. 2 CONSTANT MOVE-DISK
 6. 0 VARIABLE GOALSTACK  MAX-#DISKS 1- DUP * 2+ 8 * 2- ALLO.
 7. 0 VARIABLE GS.PTR  GOALSTACK GS.PTR !
 8. 0 VARIABLE SOURCE
 9. 0 VARIABLE TARGET
10. 0 VARIABLE #DISKS
11. 0 VARIABLE STARTED
12. 0 VARIABLE SPARE
13. \ GOALSTACK support words
14. : >GSTACK  ( n n -- n)  DUP ROT SWAP ! 2+ ;
15. : GSTACK>  ( n -- n n)  2- DUP @ ;
16. : SPARE!  ( --)  SOURCE @ TARGET @ OR 7 XOR SPARE ! ;
18. : ADDGOAL  ( --n n n n)
19.       GS.PTR @ >GSTACK >GSTACK >GSTACK >GSTACK GS.PTR ! ;
20. : REMOVEGOAL  ( --)
21.       GS.PTR @ GSTACK> DROP  GSTACK> #DISKS !
22.       GSTACK> SOURCE ! GSTACK> TARGET !  SPARE!  GS.PTR ! ;
23. : IS-GOAL  GS.PTR @ 2- @ = ;
24. : IS-#-OF-DISKS  GS.PTR @ 4 - @ = ;
25. : .PEG  DUP 1 =
26.       IF  ." A" DROP
27.       ELSE 2 =
28.         IF  ." B"
29.         ELSE  ." C"
30.         ENDIF
31.       ENDIF ;
32. \ towers rules
33. *RESET-FORPS*
34. RULE:  START-TOWERS  PRIORITY: 10
35.   *IF*     STARTED @ 0=
36.   *THEN*   GOALSTACK GS.PTR !
37.            0 0 0 0 ADDGOAL
38.            MOVE-TOWER  #DISKS @  HOME  GOAL ADDGOAL
39.            STARTED TRUE
40.   *END*
41. RULE:  MOVE-TOWER-RULE
42.   *IF*     MOVE-TOWER IS-GOAL
43.   *THEN*   REMOVEGOAL
44.            MOVE-TOWER  #DISKS @ 1-  SPARE  @  TARGET @ ADDGOAL
45.            MOVE-DISK   #DISKS @     SOURCE @  TARGET @ ADDGOAL
46.            MOVE-TOWER  #DISKS @ 1-  SOURCE @  SPARE  @ ADDGOAL
47.   *END*
48. RULE:  MOVE-SINGLE-DISK-TOWER  PRIORITY: 1
49.   *IF*     MOVE-TOWER IS-GOAL AND 1 IS-#-OF-DISKS
50.   *THEN*   REMOVEGOAL
51.            MOVE-DISK  1  SOURCE @ TARGET @ ADDGOAL
52.   *END*
53. RULE:  MOVE-SINGLE-DISK
54.   *IF*     MOVE-DISK IS-GOAL
55.   *THEN*   REMOVEGOAL
56.            TARGET @ SOURCE @ ." Move disk on peg " .PEG ."  to peg " .PEG CR
57.   *END*
```

```
58. : .HEADER  CR ." Towers of Hanoi,     Disks: " #DISKS ? CR ;
59. : ?NUM   ." Number of disks? " QUERY BL WORD HERE NUMBER DROP ;
60. : TOWERS  CR ?NUM #DISKS !  STARTED FALSE  CR .HEADER CR  FORPS  CR CR ;
```

XB FORMATTER

by Richard M. Mitchell

Many of you have inquired about how the newsletter layout is now done. Well, the specifics of what I do is surely different from what most of you could utilize because printer commands are not very standardized, but there are some tips that can apply to use of almost any dot matrix printer.

Every page of the newsletter is printed in a single pass through the printer by processing text through an XB program (with STRING MASTER used to improve execution times), a personalized formatter, similar in function to the TI-Writer Formatter.

One can tailor a formatter program to allow a computer to output to a particular printer. The formatter approach is easier, more versatile and more conducive to a positive attitude than "canned" software that must be forced to (almost) drive a printer. The formatter can utilize the same syntax structure as the TI-Writer Formatter; i.e., "dot" commands. With a combination of dot commands and built-in features for printer set-up commands that are used consistently, a formatter can be a real workhorse!

As you have likely noticed, the 80 column limit of TI-Writer is a barrier to efforts to imbed printer commands and still print 80 columns of characters. So, the example Formatter listed below provides dot command capability for italicizing (most dot matrix printers use <esc> <4> and <esc> <5> for italics on and off) particular sections of a line of text, thus allowing printing a full 80 characters of text along with imbedded italicizing command sequences! Use the dot commands as in this example (X is italics on, O is italics off):

```
.ITALICS
X            O
Italicizing is easy!
```

Note that italics are automatically turned off at the end of each line, so even the eightieth character can be italicized. As you can see, the dot commands can easily be inserted into completed text, without having to modify anything while writing!

Here's the program, a minimal implementation of a formatter to serve as an example. Note that I save my files from the TI-Writer Editor with a PF, F DSKx.FNAME for a Display, Fixed 80 file format. Also note that the program is written to allow easily adding additional dot commands into the DOT subprogram.

```
> 100 OPEN #1:"DSK1.FORMAT/X",
  DISPLAY ,FIXED 80,INPUT !041
> 110 OPEN #2:"PIO" !254
> 120 LINPUT #1:A$ !187
> 130 IF SEG$(A$,1,1)="." THEN
   CALL DOT(A$)!013
> 140 PRINT #2:A$ !174
> 150 IF EOF(1)THEN CLOSE #1 E
  LSE 120 !190
> 160 CLOSE #2 :: END !165
> 1000 SUB DOT(A$)!037
> 1010 IF SEG$(A$,2,7)="ITALIC
  S" THEN GOSUB 1100 !116
> 1020 SUBEXIT !167
> 1100 LINPUT #1:B$ :: LINPUT
  #1:A$ :: A=LEN(A$)!033
> 1110 FOR I=1 TO A !127
> 1120 IF SEG$(B$,I,1)="X" THE
  N C$=C$&CHR$(27)&"4" !212
> 1130 IF SEG$(B$,I,1)="O" THE
  N C$=C$&CHR$(27)&"5" !204
> 1140 C$=C$&SEG$(A$,I,1)!151
> 1150 NEXT I !223
> 1160 A$=C$&CHR$(27)&"5" !109
> 1170 C$="" !236
> 1180 RETURN !136
> 1190 SUBEND !168
```

BY-PASSING THE MYARC RAM DISK POWER-UP

by Mike Dodd

If you have the MYARC XB II cartridge, you have probably noticed that the RAM disk always wipes out your >6000 RAM bank unless you write protect with a Gram Kracker. The following GK patch will by-pass the power-up routine in the RAM disk, which prevents it from clearing out your >6000 bank. Now you can leave the write-protect off (e.g., to act as

a Super-cart) and not worry about it being zapped!

To make the change, enter the GRAM Kracker Memory editor. Press FCTN 1 to select GRAM, and FCTN 5 for search. Type ØØØØ for the start, and Ø3ØØ for the end. Press FCTN = for hex, then FCTN 9 to enter the search window and type 878ØDØ. Press FCTN S to back the cursor onto the "Ø" in DØ, and press ENTER. When it finds the string (mine was at gØ183), press FCTN 5 to leave the search and FCTN 9 to enter the memory field. Write down the address it is at.

Now disable write protect and type Ø519ØA. Press FCTN 9 again, use FCTN S to back over to the memory address, and type 19ØA. Press FCTN 9, ENTER to home the cursor, and type BF 8Ø DØ 11 ØØ BF 8Ø D2 4Ø Ø4 Ø5. Now take the address you wrote down and add 3 to it (>Ø183 + >ØØØ3 = >Ø186). Type that address. Turn your write protect back on, press CTRL = to leave the editor, and re-save GROM Ø to disk. To save GROM Ø, press 4 for Load/Save console, 3 for GROM Ø, and 2 for Save console. Type the filename and press ENTER. Press space (the correct GROMs are already enabled), let it finish saving, and press space again. That's all there is to it!

If you wish to run MYARC XB II, disable your write protection, change switch 2 from GRAM Ø to Op Sys, and press reset. With OP SYS selected, the patch is not in effect, so the MYARC RAM disk will execute its normal power up routine. When the title screen appears, you can re-enable GRAM Ø and proceed as normal to load MYARC XB II.

Final note: if your RAM disk is not backed up by an external power supply, you MUST run the power-up routine when you first turn the computer on. After that, if you reset the system you will not need to run the power-up routine again. You have to run it the first time, otherwise the CALL PART and CALL EMDK commands will crash. To run it without it crashing your RAM bank, disable GRAM Ø (switch to Op Sys) when you turn on the computer, making sure that the write-protect is on. When the title screen appears, enable GRAM Ø and don't worry about it again.

Editor: The above modification by-passes the power-up of whatever card is at CRU >1ØØØ. MYARC RAM disks are factory set at >1ØØØ. But, if you have changed the CRU address (for instance, for coexisting with a MYARC Personality card, which is also factory set at >1ØØØ), then the Gram Ø patch will not help the XB II situation. In that case, a circular power-up sequence beginning at >12ØØ would be required. RM

CALL CHAR AUTO-PROGRAM

by Richard M. Mitchell

The program listed below runs under STRING MASTER and writes a MERGE format program that sets up a character set based on the character set in use when this program is run. It is especially useful if you want to provide a printed listing (in a user group newsletter, for instance) of a character set. The CALL CHAR's are in console BASIC format (no multiple definitions) and trailing "Ø"'s are omitted. You may want to change the disk drive number references for your system configuration.

```
> 100 CALL INIT :: CALL LOAD("
  DSK1.SM/O")!068
> 110 OPTION BASE 1 !137
> 120 DIM A$(112),B$(112)!160
> 130 OPEN #1:"DSK1.CHARSET",D
  ISPLAY ,VARIABLE 163,OUTPUT
  !115
> 140 FOR I=1 TO 37 :: READ A
  :: LINE100$=LINE100$&CHR$(A)
  :: NEXT I :: PRINT #1:LINE10
  0$ !010
> 150 CALL LINK("SVPEEK",1024,
  0,896,A$(),8):: CALL LINK("B
  INHEX",A$(),B$()):: CALL LIN
  K("TRIM",B$(),"0")!201
> 160 FOR I=1 TO 112 :: IF LEN
  (B$(I))=0 THEN B$(I)="0" !05
  8
> 170 THISLINE$=CHR$(0)&CHR$(I
  +100)&CHR$(147)&CHR$(199)&CH
  R$(LEN(B$(I)))&B$(I)&CHR$(0)
  :: PRINT #1:THISLINE$ :: NEX
  T I !122
> 180 PRINT #1:CHR$(255)&CHR$(
  255):: CLOSE #1 !109
> 190 DATA 0,100,140,73,190,20
  0,2,51,50,177,200,3,49,52,51
  ,130,151,65,36,130,157,200,4
   !159
> 200 DATA 67,72,65,82,183,73,
  179,65,36,182,130,150,73,0 !
  063
```

COMMUNICATIONS NEWS

by Richard M. Mitchell

Fast-Term's XMODEM protocol does not always effectively interface with some host computer systems, with timeouts sometimes occurring while Fast-Term is accessing a disk. Proper implementation of XMODEM is a controversial topic that is best left to communications experts, but a simple way to skirt the issue is to set the number of XMODEM blocks to access at a time, thereby eliminating the possibility of a timeout. Good news is at hand! Read the following message, left on GEnie by Barry Traver (the file is available on GEnie, CIS, etc.):

```
**************************************
*                                    *
* FAST-TERM 1.16/2jph NOW AVAILABLE! *
*                                    *
*  Paul Charlton's Fairware program  *
*   as modified by J. Peter Hoddie   *
*                                    *
* (1) File size is shown for Xmodem  *
*  uploads _and_ downloads at start  *
*  of transfer, and record numbers   *
*  (decimal) shown during transfer.  *
* (2) A variable size Xmodem buffer  *
*  can be set for 64, 32, 16, 8, or  *
*  4 records (32 recommended for PC  *
*  Pursuit).                         *
* (3) Catalog routine can now access *
*  drives 1-9, can be paused, and    *
*  gives full information on files.  *
* (4) Files can be protected, unpro- *
*  tected, or deleted while on-line. *
* (5) Log file opens in APPEND mode  *
*  rather than OUTPUT (so that old   *
*  files will not be accidentally    *
*  overwritten).                     *
* (6) Inverse video feature has been *
*  disabled.                         *
* (7) DSRLNK routine device search   *
*  improved to work better with var- *
*  ious ramdisk configurations.      *
*                                    *
*      THIS PROGRAM IS FAIRWARE!     *
*                                    *
*      Peter's not asking for money  *
*  for his modifications, but he is  *
*  asking that if you like and use   *
*  the program and have never paid   *
*  for Fast-Term, _please_ send $15  *
*  to Paul Charlton.  (No, that's    *
*  not robbing Peter to pay Paul!)   *
*                                    *
**************************************
```

Please note that the Hoddie version of Fast-Term will not run on a MYARC 964Ø.

By the way, Barry Traver recently had surgery for a detached retina. Barry is back at the computer, but the doctors say that the full recovery period may be as long as a year. Best wishes, Barry!

For those of you who have a PC or PC compatible, I have completed XBQB.BAS, an equivalent of XB's ACCEPT AT. XBQB.BAS, a subprogram that runs in Quick BASIC 3.Ø or 4.Ø, greatly simplifies co- or cross-development of programs on the 4A and PC. The file is available in Section 12 of the TI RT on GEnie or in DL 8 of the TI FORUM on CIS. For those of you who do not access the networks, the file is available at a nominal price in the Bytemaster catalog in this issue.

Retrospective: DSK1.LOAD

by Richard M. Mitchell

A recent letter inquired as to why the loader program in the first issue of *Super 99 Monthly* always attempts to access a file named "DSK1.LOAD". The code for that is programmed into the Extended BASIC cartridge (at >6352 in GROM in one module examined) and the filename can only be changed through hardware with programmable GRAM. Perhaps the reader purchased all or a portion of his system second-hand, a common situation today, and did not receive the documentation that covered that feature of XB.

Unlike loaders that "hard-code" either the filenames available or the memory address of a filename to over-write, the *S99M* loader calculates the address of the memory location to over-write and therefore is flexible and easy to customize. The little trick involved is the use of a GOSUB in line 4ØØ to contain the address calculation in a single line of code, thereby eliminating dependence on line 4ØØ being at a specific memory location. Of course, over-writing code in memory is not, generally, a recommended programming practice, but for writing XB loaders, most alternatives are too cumbersome for the average XB programmer to tackle.

PLAY A SOUND LIST!

by Richard M. Mitchell

Well, I've been working with assembly language sound routines recently, so I decided to share what is completed. The following a/l code links to XB and will play a sound list. Unfortunately, I haven't worked out anything that I consider of adequate quality for publication for creating sound lists yet. I am working on routines that will allow far more intricate sound processing than can be accomplished in the relatively slow XB environment. For now, I have included an XB listing that will play the chimes demo from the E/A manual. Note that I have added "line numbers" to aid in working from the E/A Editor. Also, remember to omit the remarks from the XB program, those are checksums (see 12/86 issue).

```
0001  * PARMS:
0002  *  1 TO 16 PARMS OF STRINGS
0003  *    THAT ARE SOUND LISTS.
0004  *  PARMS CAN BE DIRECT STRINGS
0005  *    OR IN THE FORM A$, A$(0),
0006  *    A$(0,0), A$(), A$(,), ETC.
0007  *  ENTIRE ARRAYS, SUCH AS A$(,)
0008  *    OR A$() ARE ALLOWED, BUT
0009  *    ONLY THE FIRST ELEMENT,
0010  *    SUCH AS A$(0,0), WILL BE
0011  *    USED.
0012
0013          DEF  PLAY
0014  ARGS    EQU  >8312
0015  VSBR    EQU  >2028
0016  VMBR    EQU  >202C
0017  VSTKPT  EQU  >836E
0018  GPLWS   EQU  >83E0
0019  STABAD  EQU  >83CC
0020  STABLO  EQU  >83FD
0021  SNDBYT  EQU  >83CE
0022  ARGID   EQU  >8300
0023  ERRSNM  EQU  >0700
0024  ERR     EQU  >2034
0025
0026  PLAY    LWPI MYWS
0027          LIMI 0
0028          LI   R9,1
0029  NEXT    MOVB @ARGID-1(R9),R8
0030          SRL  R8,8
0031          MOV  R9,R3
0032          DEC  R3
0033          SLA  R3,3
0034          MOV  @VSTKPT,R0
0035          S    R3,R0
0036          CI   R8,1
0037          JEQ  NEXT3
0038          CI   R8,3
0039          JEQ  NEXT3
0040          CI   R8,5
0041          JEQ  NEXT2
0042          LI   R0,ERRSNM
0043          BLWP @ERR
0044  NEXT2   LI   R1,REG10
0045          LI   R2,2
0046          BLWP @VMBR
0047          MOV  R10,R0
0048          BLWP @VSBR
0049          SB   @OFFSTD,R1
0050          SRL  R1,8
0051          SLA  R1,1
0052          A    R1,R0
0053          AI   R0,6
0054          LI   R1,REG10
0055          BLWP @VMBR
0056          JMP  NEXT4
0057  NEXT3   AI   R0,4
0058          LI   R1,REG10
0059          LI   R2,2
0060          BLWP @VMBR
0061  NEXT4   MOV  R10,@STABAD
0062          SOCB @H01,@STABLO
0063          MOVB @H01,@SNDBYT
0064          LIMI 2
0065  LOOP    MOVB @SNDBYT,@SNDBYT
0066          JNE  LOOP
0067          INC  R9
0068          LIMI 0
0069          CB   @LSB9,@ARGS
0070          JLE  NEXT
0071  RETURN  LWPI GPLWS
0072          B    @>006A
0073
0074  MYWS    BSS  >20
0075  LSB9    EQU  MYWS+>13
0076  REG10   EQU  MYWS+>14
0077  H01     BYTE >01
0078  OFFSTD  BYTE >80
0079          END
```

```
> 100 CALL INIT :: CALL LOAD("
  DSK1.PLAY/O")!220
> 110 FOR I=1 TO 118 :: READ A
   :: A$=A$&CHR$(A):: NEXT I !
  164
> 120 CALL LINK("PLAY",A$,A$)!
  053
> 130 END !139
> 1000 DATA 5,159,191,223,255,
  227,1 !188
> 1010 DATA 9,142,1,164,2,197,
  1,144,182,211,6 !222
> 1020 DATA 3,145,183,212,5 !1
  33
> 1030 DATA 3,146,184,213,4 !1
  35
```

```
1040 DATA 5,167,4,147,176,21
4,5 !089
1050 DATA 3,148,177,215,6 !1
43
1060 DATA 3,149,178,216,7 !1
47
1070 DATA 5,202,2,150,179,20
8,6 !078
1080 DATA 3,151,180,209,5 !1
33
1090 DATA 3,152,181,210,4 !1
26
1100 DATA 5,133,3,144,182,21
1,5 !072
1110 DATA 3,145,183,212,6 !1
34
1120 DATA 3,146,184,213,7 !1
38
1130 DATA 5,164,2,147,176,21
4,6 !085
1140 DATA 3,148,177,215,5 !1
42
1150 DATA 3,149,178,216,4 !1
44
1160 DATA 5,197,1,150,179,20
8,5 !089
1170 DATA 3,151,180,209,6 !1
34
1180 DATA 3,152,181,210,7 !1
29
1190 DATA 3,159,191,223,0 !1
34
```

A/L INPUT

by Richard M. Mitchell

Someone on GEnie recently requested a minimal implementation of an assembly INPUT routine, so I quickly roughed this one out. The routine also re-displays the input on the next screen row. It uses only RØ, R1 and R2, so it could likely be accessed by BL or BLWP with little modification. Note that it was written for E/A (REF is used).

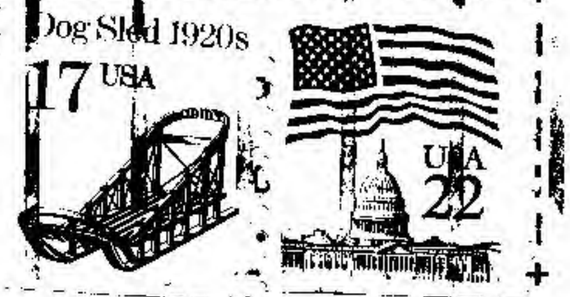
```
0001          DEF  INPUT
0002          REF  KSCAN
0003  VDPWA   EQU  >8C02
0004  VDPWD   EQU  >8C00
0005  KEYBRD  EQU  >8374
0006  KEYCOD  EQU  >8375
0007  STATUS  EQU  >837C
0008  MYWS    EQU  >8300
0009  R1LB    EQU  MYWS+3
0010  R2LB    EQU  MYWS+5
0011  TABSIZ  EQU  5
0012
0013  INPUT   MOV  R11,@STRRET
0014          LWPI MYWS
0015          MOV  @SIZE,R1
0016          MOV  R1,R0
0017          DEC  R0
0018  CLRBUF  DEC  R1
0019          MOVB @BLANK,@BUFFER(R1)
0020          MOV  R1,R1
0021          JNE  CLRBUF
0022          MOVB @KEYBRD,@STRBRD
0023          CLR  @KEYBRD
0024  LOOP    CB   @CURSOR,@CURSWP
0025          JEQ  SWAP1
0026          MOVB @CURSOR,@CURSWP
0027          JMP  SWAP2
0028  SWAP1   MOVB @BUFFER(R1),@CURSWP
0029  SWAP2   BL   @PUTCH2
0030          LI   R2,120
0031  CKSCAN  BLWP @KSCAN
0032          CB   @KEYCOD,@CHRTAB
0033          JEQ  TSTTAB
0034          CB   @KEYCOD,@CHRTAB+1
0035          JEQ  TSTTAB
0036          CB   @BLANK,@STATUS
0037          JEQ  TSTTAB
0038          DEC  R2
0039          JNE  CKSCAN
0040  TSTTAB  LI   R2,>2200
0041  DELAY2  DEC  R2
0042          JNE  DELAY2
0043  CKTAB   CB   @KEYCOD,@CHRTAB(R2)
0044          JLE  LOKTAB
0045          INC  R2
0046          CI   R2,TABSIZ
0047          JNE  CKTAB
0048          JMP  LOOP
0049  LOKTAB  SLA  R2,1
0050          AI   R2,$+6
0051          B    *R2
0052          JMP  LEFT
0053          JMP  RIGHT
0054          JMP  RETURN
0055          JMP  LOOP
0056          JMP  ASCII
0057  LEFT    CB   @KEYCOD,@CHRTAB
0058          JNE  LOOP
0059          MOV  R1,R1
0060          JEQ  LOOP
0061          BL   @PUTCHR
0062          DEC  R1
0063          JMP  LOOP
0064  RIGHT   C    R1,R0
0065          JEQ  LOOP
0066          BL   @PUTCHR
0067          INC  R1
0068          JMP  LOOP
0069  ASCII   MOVB @KEYCOD,@BUFFER(R1)
0070          BL   @PUTCHR
0071          C    R1,R0
0072          JEQ  LOOP
0073          INC  R1
```

```
0074            JMP   LOOP
0075  RETURN    CB    @KEYCOD,@CHRTAB+2
0076            JNE   LOOP
0077            BL    @PUTCHR
0078            MOV   @SIZE,R1
0079  RET1      CB    @BUFFER-1(R1),@BLANK
0080            JNE   RET2
0081            DEC   R1
0082            JNE   RET1
0083  RET2      MOVB  @R1LB,@BUFLEN
0084            MOV   R1,R0
0085            CLR   R1
0086            MOV   @SCRLOC,R2
0087            AI    R2,32
0088            ORI   R2,>4000
0089            MOVB  @R2LB,@VDPWA
0090            MOVB  R2,@VDPWA
0091  SHWBUF    C     R1,R0
0092            JEQ   SHWBFE
0093            MOVB  @BUFFER(R1),@VDPWD
0094            INC   R1
0095            JMP   SHWBUF
0096  SHWBFE    SETO  R1
0097  DELAY     DEC   R1
0098            JNE   DELAY
0099            MOVB  @STRBRD,@KEYBRD
0100            CLR   R0
0101            MOVB  R0,@STATUS
0102            MOV   @STRRET,R11
0103            RT
0104
0105  PUTCHR    MOVB  @BUFFER(R1),@CURSWP
0106  PUTCH2    MOV   R1,R2
0107            A     @SCRLOC,R2
0108            ORI   R2,>4000
0109            MOVB  @R2LB,@VDPWA
0110            MOVB  R2,@VDPWA
0111            MOVB  @CURSWP,@VDPWD
0112            RT
0113
0114  BUFLEN    BYTE  0
0115  BUFFER    BSS   28
0116  CURSOR    BYTE  30
0117  CURSWP    BYTE  30
0118  BLANK     BYTE  32
0119  STRBRD    BYTE  0
0120  CHRTAB    BYTE  8,9,13,31,126
0121  SIZE      DATA  28
0122  SCRLOC    DATA  2
0123  STRRET    DATA  0
0124            END
```

The Smart Programmer is published by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663. All correspondence received will be considered unconditionally assigned for publication and copyright and subject to editing and comments by the Editor of *The Smart Programmer*.

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663
U.S.A.

FIRST CLASS MAIL



Postmaster: Address Correction Requested

FIRST CLASS MAIL